Наследование классов

Класс, который наследуется другим классом, называется базовым классом. Иногда его также называют родительским классом. Класс, выполняющий наследование, называется производным классом или потомком. Следуя традиции, будем пользоваться терминами «базовый класс» и «производный класс».

Спецификаторы доступа

В C++ члены класса классифицируются в соответствии с правами доступа на следующие три категории: публичные (public), частные (private) и защищенные (protected). Любая функция программы имеет доступ к публичным членам. Доступ к частному члену имеют только функции-члены класса или функции-друзья класса. Защищенные члены аналогичны частным членам. Разница между ними появляется только при наследовании классов.

Когда один класс наследует другой, все публичные члены базового класса становятся публичными членами производного класса. В противоположность этому частные члены базового класса не доступны внутри производного класса. Например, рассмотрим следующий фрагмент:

```
class X {
int i;
int j;
public:
void get_ij();
void put_ij();
};
class Y: public X {
int k;
public:
int get_k();
void make_k();
};
```

Класс Y наследует и имеет доступ к публичным функциям get_ij() и put_ij() класса X, но не имеет доступа к i и j, поскольку они являются частными членами X. Во всех случаях частные члены остаются частными, т. е. доступными только в том классе, в котором они объявлены. Таким образом, частные члены не могут участвовать в наследовании.

В связи с тем, что частные члены не могут быть наследованы, возникает интересный вопрос: что если необходимо оставить член частным и вместе с тем позволить использовать его производным классам? Для таких целей имеется другое ключевое слово — protected (защищенный). Защищенный член подобен частному, за исключением механизма наследования. При наследовании защищенного члена производный класс также имеет к нему доступ. Таким образом, указав спецификатор доступа protected, можно позволить использовать член внутри иерархии классов и запретить доступ к нему извне этой иерархии. например:

```
class X {
protected:
int i;
int j;
public:
void get_ij();
void put_ij();
```

```
};
class Y: public X {
int k;
public:
int get_k();
void make_k();
};
```

Здесь класс Y имеет доступ к i и j, и в то же время они остаются недоступными для остальной части программы. Когда элемент объявляется защищенным, доступ к нему ограничивается, но вместе с тем можно наследовать права доступа. В отличие от этого в случае частных членов доступ не наследуется.

Другой важной особенностью ключевых слов private, protected и public служит то, что они могут появляться в объявлении в любом порядке и в любом количестве.

Например, следующий код является вполне законным:

```
class my_class {
protected:
int i;
int j;
public:
void f1();
void f2();
protected:
int a;
public:
int b;
};
```

Спецификатор доступа при наследовании базового класса

От того, с каким спецификатором доступа объявляется наследование базового класса, зависит статус доступа к членам производного класса. Общая форма наследования классов имеет следующий вид:

```
class имя_класса: доступ имя_класса {
....
};
```

Здесь доступ определяет, каким способом наследуется базовый класс. Спецификатор доступ может принимать три значения — private, public и protected. В случае, если спецификатор доступ опущен, то по умолчанию подразумевается на его месте спецификатор public. Если спецификатор доступ принимает значение public, то все публичные и защищенные члены базового класса становятся соответственно публичными и защищенными членами производного класса. Если спецификатор доступ имеет значение private, то все публичные и защищенные члены базового класса становятся частными членами производного класса. Если спецификатор доступ принимает значение protected, то все публичные и защищенные члены базового класса становятся защищенными членами производного класса. Для того чтобы усвоить все эти преобразования, рассмотрим пример:

```
#include <iostream.h>
class X {
protected:
int i;
int j;
public:
```

```
void get_ij () {
cout << "Enter two numbers: ";
cin >> i >> j;
}
void put_ij() { cout << i << " " << j << "\n"; }
};
// в классе Y, i и j класса X становятся защищенными членами
class Y: public X {
int k;
public:
int get_k() { return k; }
void make_k() { k = i*j; }
};
/* класс Z имеет доступ к i и j класса X, но не к k класса Y, поскольку он является
частным */
class Z: public Y {
public:
void f();
};
// i и j доступны отсюда
void z::f()
{
i = 2; // нормально
j = 3; // нормально
}
int main()
{
Y var;
Z var2;
var.get_ij();
var.put_ij();
var.make_k();
cout << var.get_k();
cout << "\n";
var2.f();
var2.put_ij();
return 0;
}
```

Поскольку класс Y наследует класс X со спецификатором доступа public, то защищенные элементы класса X становятся защищенными элементами класса Y. Это означает, что они могут далее наследоваться классом Z, и эта программа будет откомпилирована и выполнена корректно. Однако, если изменить статус X при объявлении Y на private, то, как доказано в следующей программе, класс Z не имеет права доступа к i и j и к функциям get_ij() и put_ij(), поскольку они стали частными членами Y:

```
#include <iostream.h>
class X {
protected:
int i;
int j;
public:
void get_ij(){
cout << "Enter two numbers: ";
```

```
        cin >> i >> j;
        }
        void put_ij() { cout << i << " " << j << "\n"; }
        };
        // теперь i и j преобразованы к частным членам Y
        class Y: private X {
        int k;
        public:
        int get_k() { return k; }
        void make_k() { k = i*j; }
        };
        /* поскольку i и j частные для Y, они не могут наследоваться в Z */
        class Z: public Y {
        public:
        void f();
        };
// теперь данная функция не работает
        void Z::f()
        {
        // i = 2; i и j больше не доступны
        // j = 3;
        }
int main()
        {
Y var;
        Z var2;
        // var.get_ij(); больше не доступна
        // var.put_i j(); больше не доступна
        var .make_k();
        cout << var.get_k();
        cout << " \n";
        var2.f();
        // var2.put_ij(); больше не доступна
        return 0;
        }
```

Когда при объявлении класса Y перед базовым классом X имеется спецификатор доступа private, члены i, j, get_ij() и put_ij() становятся частными членами Y и поэтому не могут наследоваться классом Z, так что класс Z не имеет больше к ним доступа.

При использовании производных классов важно представлять себе, каким образом и когда исполняются конструкторы и деструкторы базового и производного классов. Начнем рассмотрение с конструкторов.

Как базовый класс, так и производный класс могут иметь конструкторы. (В многоуровневой иерархии классов каждый из классов может иметь конструкторы, но мы начинаем с наиболее простого случая.) Когда базовый класс имеет конструктор, этот конструктор исполняется перед конструктором производного класса. Например, рассмотрим следующую короткую программу:

```
        #include <iostream.h>
        class Base {
        public:
        Base () {cout << "\nBase created\n";}
        };
        class D_class1(): public Base {
```

```
public:
D_class1() {cout << "D_class1 created\n";}
};
int main()
{
D_class1 d1;
// ничего не делается, но выполняются конструкторы
return 0;
}
```

Эта программа создает объект типа D_class1. Она выводит на экран следующий текст:

```
Base created
D_class1 created
```

Здесь d1 является объектом типа D_class1, производным класса Base. Таким образом, при создании объекта d1 сначала вызывается конструктор Base(), а затем вызывается конструктор D_class1().

Конструкторы вызываются в том же самом порядке, в каком классы следуют один за другим в иерархии классов. Поскольку базовый класс ничего не знает про свои производные классы, то его инициализация может быть отделена от инициализации производных классов и производится до их создания, так что конструктор базового класса вызывается перед вызовом конструктора производного класса.

В противоположность этому деструктор производного класса вызывается перед деструктором базового класса. Причину этого также легко понять. Поскольку уничтожение объекта базового класса влечет за собой уничтожение и объекта производного класса, то деструктор производного объекта должен выполняться перед деструктором базового объекта. Следующая программа иллюстрирует порядок, в котором выполняются конструкторы и деструкторы:

```
#include <iostream.h>
class Base {
public:
Base() {cout << "\nBase created\n";}
~Base() {cout << "Base destroyed\n\n,"; }
};
class D_class1(): public Base {
public:
D_class1() {cout << "D_class1 created\n";}
~D_class1() {cout << "D_class1 destroyed\n "; }
};
int main()
{
D_class1 d1;
cout << " \n";
```

```
return 0;
}
```

Эта программа выдаст следующий текст на экран:

```
Base created
D_class1 created
D_class1 destroyed
Base destroyed
```

Производный класс может служить базовым классом для создания следующего производного класса. Когда такое происходит, конструкторы исполняются в порядке наследования, а деструкторы — в обратном порядке. В качестве примера рассмотрим следующую программу, использующую класс D_class2, производный от класса D_ciass1:

```
#include <iostream.h>
class Base {
public:
Base() {cout << "\nBase created\n";}
~Base() {cout << "Base destroyed\n\n";}
};
class D_class1() : public Base {
public:
D_class1() {cout << "D_class1 created\n";}
~D_class1() {cout << "D_class1 destroyed\n ";}
};
class D_class2: public D_class1 {
public:
D_class2() {cout << "D_class2 created\n";}
~D_class2() {cout << "D_class2 destroyed\n ";}
};
int main()
{
D_class1 d1;
D_class2 d2;
cout << "\n";
return 0;
}
```

Эта программа выдаст следующий результат:

```
Base created
D_class1 created
Base created
D_class1 created
D_class2 created
D_class2 destroyed
D_class1 destroyed
Base destroyed
D_class1 destroyed
Base destroyed
```